LA-UR-19-29589
Approved for public release; distribution is unlimited.

Title: Open MPI Testing Infrastructure Update - September, 2019

Author(s): Pritchard, Howard Porter Jr.
Gutierrez, Samuel Keith
Rezanka, Deborah Suzanne
Topa, Daniel Martin

Intended for: Report

Issued: 2019-09-24

Open MPI Testing Infrastructure Update- September, 2019

Deborah Rezanka (LANL), Dan Topa (LANL), Samuel Gutierrez, Howard Pritchard 9/19

This report serves as part the deliverable for ECP OMPI-X Story STPM13-56. Other deliverables include code and documentation checked into various repos on GitHub.

MTT Python Client and Related Infrastructure Support

Since the last update on the Open MPI regression testing infrastructure, and the deployment of the MTT Python client on more DOE HPC centers, problems with the client and its interaction with the community PostGreSQL database server at AWS began to appear. In particular, problems were found with the reporting of failed and timed out tests to the community database. Successes were being properly reported, but failures and timeouts were not. This proved to be a very difficult problem to debug, as it was not obvious where in the various components of the MTT reporting infrastructure – the MTT client, the intermediate Cherrypy server, or the backend database server – the problem was occurring. As part of the effort to resolve this problem, a local MTT database server and intermediate Cherrypy server were set up and tested. Eventually the problem was determined to be in the MTT Python client. The client plugins handling test launch using various job launchers (mpirun, srun, aprun) had to be refactored to fix this reporting problem. In the course of setting up this local version of the Open MPI community's MTT infrastructure, numerous problems were found with the existing MTT documentation. The lack of reliable documentation complicated setting up a local version of the MTT infrastructure. These problems were largely resolved on the MTT project's Github wiki pages (https://github.com/open-mpi/mtt/wiki) and INSTALL writeups in the MTT GitHub repo. Issues were opened to track remaining documentation deficiencies related to setting up a MTT database instance.

Other work on the MTT Python client included reworking how the test input files are specified in the job launcher plugins and a new plugin was added to run PMIx unit tests.

The MTT Cherrypy server code was futurized to work both with Python2 and Python3.

The PostGreSQL database schema template was updated to remove fields that are no longer used. These fields were sometimes causing issues with inserting test results reported by the Python client into the database.

bueno: A Software Framework for Automated, Reproducible Benchmarking and Analysis

Although the Open MPI MTT infrastructure has proven to be quite effective for running MPI tests (and tests for other products such as PMIx) to check for regressions, it has not proven to be as useful for testing Open MPI against more complex applications. Often such applications require special setups, input files, etc. that would require writing customized launcher plugins for individual applications. In addition, when archiving application performance metrics, there needs to be a reliable way to record the provenance of the runs – compilers used, versions of external dependencies used, possibly other characteristics of the runtime environment, etc. which would be impractical to record in an easily accessible way in the MTT PostGreSQL database. As a consequence, work began this past quarter on a new, lightweight infrastructure targeting this kind of application-centric Open MPI testing – Bueno.

System benchmarking provides a means to compare or assess the performance of hardware or software against a point of reference. Because of the multitude of factors that ultimately influence a benchmark's results, reproducibility is challenging. To aid in this, we present an extensible software framework named *bueno* that provides mechanisms to record and automate many arduous, error-prone benchmarking tasks: environmental discovery, environmental setup, program compilation, program execution, data storage, and analysis. In this report, we summarize bueno's software architecture, current feature set, and methodology for supporting automated, reproducible benchmarking and analysis.

bueno is a small (~1,256 SLOC) Python-based (Python 3.5) software framework with minimal external software dependencies. Its internal software architecture consists of *services*, which are accessible through a command-line interface (CLI), and a collection of public Python modules made available to Python programs executed by bueno's *run* service. The remainder of this section describes both the CLI and public module services provided by bueno.

Two CLI services are currently available through bueno: *build* and *run*.

The *build* service is a front-end to container builder backends (e.g., Docker, Charliecloud). This service acts as an abstraction layer that hides the steps required to build a container image given a particular backend and specification (e.g., a Dockerfile). Additionally, the build service annotates the generated container images with embedded metadata detailing their build specification, environment, and process. We posit that enough metadata are stored within the generated images to perform operations critical to reproducibility and post-mortem analysis.

The *run* service coordinates container image activation and the execution of *bueno run scripts*—a programmatic description of the steps required to conduct a benchmarking activity. Currently,

there are two image activators implemented in bueno: *charliecloud* and *none*. The former uses Charliecloud to activate a given container image and the latter is a pass-through to the host.

Because of the diversity among computer platforms, system software, and programs of interest, the running of programs and subsequent analysis of their generated outputs is expressed through Python programs executed by bueno's *run* service. A collection of public Python modules is made available to these programs that aid in conducting benchmarking activities. Examples include: command dispatch to the host or container, logging, metadata asset agglomeration, concise expression of structured experiment generation, and programmable pre- and post-experiment actions.

bueno is currently undergoing review for release as an open-source project. Once complete, bueno's source will be made publicly available on GitHub under the BSD license.

Spack Builds of Open MPI

The test matrix of regular builds of Open MPI via Spack was expanded to include additional IBM (LLNL RZansel system) and ARM based platforms. During this time period separate testing of PMIx builds on via Spack was also started. Continued updating the Open MPI and PMIx Spack recipes with new versions.